

# JMFL RESZAZO

software craftsman & digital creative



# **Arquitectura de Microservicios con DAPR en .NET**



**Bienvenidos**  
**Acerca de...**



**Jose María Flores Zazo, autor**

*¡Hola! Gracias por entrar en  
“Arquitectura de Microservicios con DAPR”.  
Espero poder aportarte los conocimientos mínimos y necesarios  
para que puedas ponerlo en práctica.*



# Introducción

## Preámbulo

Esta es la segunda parte de un ciclo de tres workshops dedicados a Microservicios, donde:

- I. **Arquitectura de Microservicios con contenedores usando .NET5**, hablaba del Proyecto Tye y daba ciertas explicaciones sobre que son los microservicios, así como varias tecnológicas a parte de los microservicios, tales como gRPC o WSL. Si no habéis realizado ese workshop el contenido lo puedes encontrar en:
  - <https://github.com/jmfloreszazo/microservicesapp>
  - <https://speakerdeck.com/jmfloreszazo/arquitectura-de-microservicios-con-contenedores-usando-net5>
- II. **Arquitectura de Microservicios con DARP en .NET**, donde te encuentras actualmente, trataremos de esta tecnología que nos permitirá trabajar con componentes distribuidos. Además de tocar Azure APIM para ahondar un poco en Arquitectura propiamente dicha.
- III. **Arquitectura de Microservicios: Project Tye y DARP, codo con codo** (en preparación), donde vamos a ver la simbiosis existente entre estas dos tecnologías. Puedes realizar los anteriores workshops en el orden que desees, pero cuando llegues a este, recomiendo haber realizado los dos anteriores, aunque mi recomendación es seguir el orden que he puesto.

Con estos tres workshops, dispondrás de una visión de herramientas y tecnologías que son el futuro de los microservicios.



# Microservicios

## Conceptos

Mi intención es darte una idea del desarrollo de aplicaciones cloud-native. Y los microservicios son 100% cloud-native.

Voy a suponer que ya sabe que son los microservicios y nos vamos a centrar solamente en algunos aspectos técnicos.

Estos aspectos técnicos de los microservicios, definidos en pocas palabras, tienden a depender de los siguientes patrones y tecnologías:

- **Sidecar**: el patrón Sidecar es uno de los patrones más utilizados en el mundo de los contenedores. Las propias mallas de servicios (services meshes) aprovechan este patrón, al igual que la mayoría de las soluciones de este ecosistema.
- **PUB/SUB**: Establecen la comunicación asíncrona entre servicios.
- **gRPC** y **REST HTTP/2**, aceleran la comunicación directa entre servicios.
- **API gateways**: exponen algunos servicios al mundo exterior.

DARPA nos permitirá utilizar todos los patrones y tecnologías que hemos enumerado anteriormente.



# Microservicios

## Introducción

### DAPR – Distributed Application Runtime

Cuando trabajamos con microservicios DAPR nos resultará muy útil ya que DAPR es un runtime de aplicaciones que nos ayuda a trabajar con componentes distribuidos.

A principios de 2021 se publicó la versión 1.0. Por tanto, estamos ante una tecnología relativamente nueva.

DAPR tiene una gran afinidad con los microservicios y concretamente con K8s.

La propuesta de DAPR es ser una capa de abstracción entre código de la aplicación y el ecosistema subyacente con los que interactúa el código. Desacopla totalmente el código de su destino y permite sustituir sin problemas un destino por otro, sin cambiar una sola línea de código.

DAPR se ocupa de forma nativa de las tecnologías y los patrones anteriormente citados. DAPR también trabaja con componentes.

La documentación podrás encontrarla en: <https://docs.dapr.io/>



# DARP

## Componentes (1/2)

Puedes ver todos los componentes en: <https://docs.dapr.io/concepts/components-concept/>

	BINDINGS	STATE	PUB/SUB	SECRETS
Azure	Cosmos DB, Storages, Event Grid, Event Hub, Signal R, ...	Storages, Comos DB, SQL Server.	Event Hub, Service Bus, ...	Azure Key Vault
AWS	Dynamo DB, Kinesis, S3, SNS, SQS.	Dynamo DB	SNS/SQS	Secret Manager
GCP	Bucket, PUBSUB.	Firestore	PUBSUB	Secret Manager
Miscelaneous	Redis, Kafka, HTTP, K8s, MQTT, RabbitMQ, ...	Cassandra, Postgre, Zookeeper, Redis, MongoDB, CouchBase, ...	Kafka, MQTT, NATS, Redis, RabbitMQ, ...	Local, K8s, Hashicorp Vault, ...

Existe los componentes: **Middleware** y **Service Discovery name resolution**, que no hemos tratado en la tabla anterior. Así como he abreviado Secrets y realmente son **Secrets Stores**.



# DARP

## Componentes (2/2)

Veamos que son los componentes:

- **SECRETS:** DARP admite muchos almacenamientos de secretos. Para hacer referencia a un secreto, debemos hacer una consulta `http://localhost:3500/v1.0/secrets/{component-name}/{secret-name}`
- **PUB/SUB:** DARP soporta Azure Event Hub y Service Bus, así como Redis, RabbitMQ, Kafka o muchos más. Para publicar un mensaje en un tópico de Service Bus, se debe hacer una llamada `http://localhost:3500/v1.0/publish/{component-name}/{topic}`, mientras que suscribirse sería: `http://localhost:3500/v1.0/publish/{component-name}/{topic}`
- **BINDINGS:** DARP puede estar vinculado a diferentes almacenamientos, Cosmos DB, Azure Storages, por mencionar alguno. Y con una simple consulta `http://localhost:3500/v1.0/bindings/{bind-name}` es suficiente para enlazar uno de estos almacenamientos.
- **STATE:** DARP puede escribir pares Key/Values en todos los almacenamientos permitidos. Apuntado a `http://localhost:3500/v1.0/state/{component-name}/{key-name}`

Además, proporciona dos endpoints, que no están representados como componentes y no he puesto en el diagrama anterior, pero que muy importantes:

- **INVOKE/{APPLICATION}/METHOD/{METHOD}:** Usado para invocar directamente a un servicio desde otro servicio.
- **INVOKE/WORKFLOWS/METHOD/{WORKFLOW}:** Usado para invocar un workflow. DARP se integra con Azure Logic Apps para ejecutar workflows auto hospedados para utilizar la potencia de las Logic Apps.



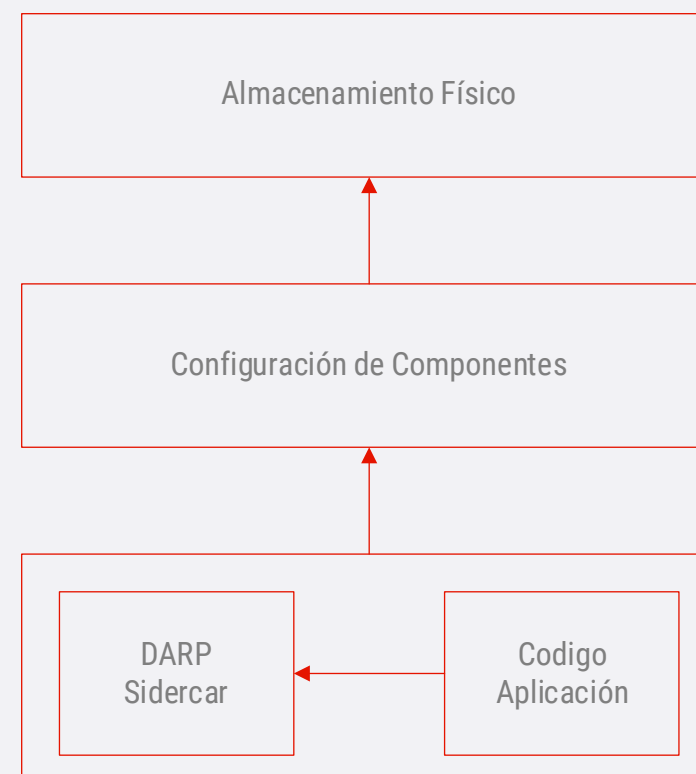


# DARP

## Entendamos los componentes <sup>(1/2)</sup>

DARP utiliza componentes para abstraer los diversos datos y almacenes de mensajes con los que interactúan las aplicaciones de cliente.

En la siguiente imagen mostramos como el contenedor sidecar\* lee la configuración del componente para eventualmente apuntar a los almacenamientos físicos:



\* El patrón sidecar, se trata al final del workshop.



# DARP

## Entendamos los componentes (2/2)

Nuestro código de la aplicación no sabe si esta hablando contra un Azure Storage o un Amazon S3. La aplicación simplemente habla con un Endpoint, los que vimos anteriormente. Depende del contenedor Sidecar encontrar el componente adecuado.

El beneficio de este enfoque es una gran simplificación del código de la aplicación, pero también significa un desacoplamiento del código con el almacenamiento con el cual debe interactuar.

Debido a esta arquitectura, podemos cambiar fácilmente de almacenamientos físicos simplemente cambiando la configuración del componente sin afectar al código de la aplicación.

Un componente es:














```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: {name}
  namespace: {K8s namespace}
spec:
  type: {component-type}
  metadata:
    - {component-specific}
```

Cuando trabaja con K8s la definición del recurso será Component y los metadatos varían según el tipo de componente. Una vez implementado, si usas AKS, DARP detectará su presencia al iniciar el contenedor Sidecar.

No te preocupes si no tienes K8s o un AKS, nosotros vamos a ejecutar los ejemplos y aprender DARP sin usar K8s.



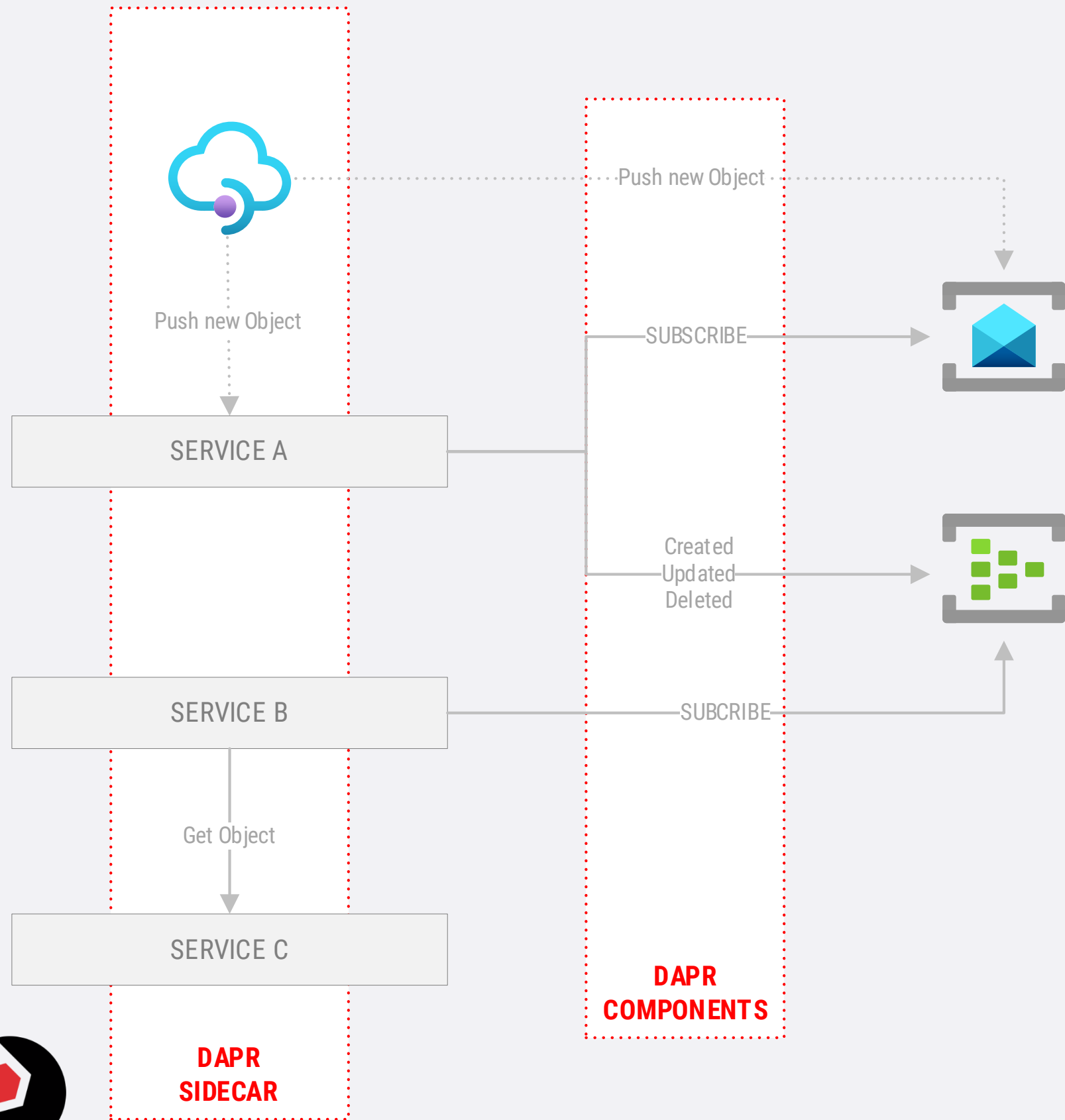
El código de la aplicación solo llama a los endpoints de DARP que corresponden a los componentes DARP, es decir, que DARP es compatible con cualquier lenguaje de programación. En nuestro caso vamos a usar el SDK de .NET Core:

	<b>Dapr.Client</b>  by dapr.io, 89,3K downloads	v1.0.0
This package contains the reference assemblies for developing services using Dapr.		
	<b>Dapr.AspNetCore</b>  by dapr.io, 78,2K downloads	v1.0.0
This package contains the reference assemblies for developing services using Dapr and ASP.NET Core.		
	<b>Dapr.Actors</b>  by dapr.io, 38,9K downloads	v1.0.0
This package contains the reference assemblies for developing Actor services using Dapr.		
	<b>Dapr.Actors.AspNetCore</b>  by dapr.io, 32,2K downloads	v1.0.0
This package contains the reference assemblies for developing Actor services using Dapr and ASP.NET Core.		
	<b>Dapr.WebSockets</b> by Reuben Bond, 17,7K downloads	v0.2.7
Portable Class Library for WebSockets, wrapping platform-specific APIs.		
	<b>Dapr.Extensions.Configuration</b>  by dapr.io, 1,66K downloads	v1.0.0
Dapr Secret Store configuration provider implementation for Microsoft.Extensions.Configuration.		
	<b>Dapr.AzureFunctions.Extension</b>  by dapr.io, 1,07K downloads	v0.10.0- preview01
This extension adds bindings for Dapr		
Prerelease		



# Ejemplo E2E

## Arquitectura del ejemplo



## Tenemos tres servicios:

- **SERVICE A:** recibe llamadas por dos canales. Uno es un tópico de Azure Service Bus y el otro una solicitud HTTP directa desde un endpoint. Cuando el servicio cree un objeto se publicará un evento Created en el Azure Event Hub. Las llamadas directas al Service A son transferidas por DARP Sidecar.
- **SERVICE B:** es un suscriptor de Event Hub y será notificado cada vez que un objeto llega al Event Hub. A su vez llamará al Service C para verificar que el objeto existe y obtener información adicional del objeto. A continuación, iniciará/modificará/cancelará el proceso en relación al evento recibido. Las llamadas directas al Service C son transferidas por DARP Sidecar.
- **SERVICE C:** simplemente devuelve un objeto, en nuestro caso alojado en memoria en tiempo de ejecución.

Con este ejemplo ya tenemos suficientes elementos e iteraciones, para realizar una implementación real con DARP.



# Ejemplo E2E

## Infraestructura de la solución (1/4)

### BICEP – Next generation of ARM Templates

Tal y como realizaba en la primera parte de esta serie de workshop, vamos a estudiar más tecnología que la que tomamos como base.

Bicep es un DSL (Domain Specific Language) para desplegar recursos en Azure de forma declarativa.

En el momento de la publicación de este workshop el desarrollo del producto esta en una fase inicial, pero con soporte por parte de Azure.

El propósito de Azure Bicep el siguiente:

- Usar un lenguaje más amigable para un desarrollador que los JSON de ARM.
- Producir una sola plantilla de ARM para evitar el uso de una cuenta de almacenamiento o cualquier otro sistema para almacenar plantillas vinculadas.

A diferencia de [Terraform](#) o [Pulumi](#), Bicep es específico de Azure.

Si ya tienes experiencia con Terraform o con Pulumi, podrás observar que el propósito de Bicep es similar a las ventajas que tenemos con Terraform o Pulumi:

- Menor número de líneas de código necesarias para crear un recurso en Azure que usando ARM.
- Usan lenguajes o bien propios como Terraform (HCL) o bien el casi más te guste con Pulumi (puedes usar C#, JS, TS, Go ...). Desde mi punto de vista esta opción es muy interesante y otorga una ventaja sobre Terraform y Bicep.

El proyecto Bicep está alojado en: <https://github.com/Azure/bicesp>



# Ejemplo E2E

## Infraestructura de la solución (2/4)

¿Qué necesitamos para poder ejecutar nuestra infraestructura?:

1. Instalar la herramienta CLI de Bicep (seguir las instrucciones: <https://github.com/Azure/bicep/blob/main/docs/installing.md>).
2. Instalar la extensión de Visual Studio Code para Bicep. Es un paso opcional, pero mejora mucho nuestro trabajo con Bicep.
3. Instalar la última versión de Azure CLI (seguir las instrucciones: <https://docs.microsoft.com/es-es/cli/azure/install-azure-cli>).
4. Abrir el fichero iac.bicep en VS Code. Lo podrás encontrar en el repo de GitHub:

<https://github.com/jmfloreszazo/microservicesappdapr>

```
iac.bicep x
iac.bicep > ...
1 // Step 1: az login
2 // Step 2: az group create --name [myResourceGroup] --location "Central US"
3
4 param location string = resourceGroup().location
5 param namespaceName string
6
7 var serviceBusNamespaceName = concat(namespaceName, '-darpsample')
8 var hubNamespaceName = concat(namespaceName, '-', uniqueString(resourceGroup().id))
9 var storageAccountName = concat(namespaceName, uniqueString(resourceGroup().id))
10
11 resource namespace 'Microsoft.EventHub/namespaces@2017-04-01' = {
12   name: '${hubNamespaceName}'
13   location: location
14   sku: {
15     name: 'Standard'
16     tier: 'Standard'
17     capacity: 1
18   }
19   properties: {}
20 }
21
22 resource eventHub 'Microsoft.EventHub/namespaces/eventhubs@2017-04-01' = {
23   name: '${namespaceName}/dapr-eh'
24   properties: {}
25 }
26
27 resource consumerGroup 'Microsoft.EventHub/namespaces/eventhubs/consumergroups@2017-04-01' = {
28   name: '${eventHub.name}/service-b'
29   properties: {}
30 }
31
32 resource serviceBusNamespace 'Microsoft.ServiceBus/namespaces@2018-01-01-preview' = {
33   name: '${serviceBusNamespaceName}'
34   location: location
35   sku: {
36     name: 'Standard'
37     tier: 'Standard'
38   }
39   properties: {}
40 }
41
42 resource serviceBusQueue 'Microsoft.ServiceBus/namespaces/topics@2017-04-01' = {
43   name: '${serviceBusNamespaceName}/dapr'
44 }
45
46 resource sa 'Microsoft.Storage/storageAccounts@2019-06-01' = {
47   name: storageAccountName
48   location: location
49   sku: {
50     name: 'Standard_LRS'
51     tier: 'Standard'
52   }
53   kind: 'StorageV2'
54   properties: {
55     accessTier: 'Hot'
56   }
57 }
58
59 resource container 'Microsoft.Storage/storageAccounts/blobServices/containers@2019-06-01' = {
60   name: '${sa.name}/default/darpsample'
61 }
62
```



# Ejemplo E2E

## Infraestructura de la solución (3/4)

La imagen muestra muy pocas líneas, esto es gracias al DSL, sin embargo, si exportas el fichero a JSON para obtener el ARM, verás que el numero de líneas crece considerablemente por tanto es:

- Más conciso
- Más legible.

Ahora llega el momento de crear la infraestructura, sigamos los siguientes pasos:

1. Instalar la herramienta CLI de Bicep (seguir las instrucciones: <https://github.com/Azure/bicep/blob/main/docs/installing.md>).
2. Ejecutar:

```
TERMINAL  PROBLEMS  1  OUTPUT  DEBUG CONSOLE
PS C:\temp\microservicesappdapr> bicep build iac.bicep
PS C:\temp\microservicesappdapr> 
```

3. Al haber generado el JSON con el comando anterior, solo, nos queda desplegar el ARM exportado para desplegar la infra:

```
TERMINAL  PROBLEMS  1  OUTPUT  DEBUG CONSOLE  1: python
PS C:\temp\microservicesappdapr> az deployment group create --resource-group 'jmfzdaprsample' --template-file iac.json --confirm-with-what-if
Please provide string value for 'namespaceName' (? for help): jmfz
- Running ..
```



# Ejemplo E2E

## Infraestructura de la solución (4/4)

Tras esperar el tiempo necesario, ya tenemos nuestra infraestructura:

jmfzdaprsample

Resource group

Search (Ctrl+/,)

«

+ Add

≡ Edit columns

🗑 Delete resource group

🔄 Refresh

📄 Export to CSV

🔗 Open query

🏷 Assign tags

➡ Move

🗑 Delete

📄 Export template

💡 Feedback

📄 Open

Overview

Activity log

Access control (IAM)

Tags

Events

Settings

Deployments

Security

Policies

Properties

Essentials

Subscription (change) : Visual Studio Enterprise – MPN

Deployments : 1 Succeeded

Subscription ID : b7612533-769e-4291-8899-da2151cdba10

Location : Central US

Tags (change) : Click here to add tags

Filter for any field...

Type == all

Location == all

Add filter

Showing 1 to 3 of 3 records.

Show hidden types

No grouping

<input type="checkbox"/> Name ↑↓	Type ↑↓	Location ↑↓
<input type="checkbox"/> jmfmz-35mlw7mi7jcoa	Event Hubs Namespace	Central US
<input type="checkbox"/> jmfmz-darpsample	Service Bus Namespace	Central US
<input type="checkbox"/> jmfmz35mlw7mi7jcoa	Storage account	Central US

Home > jmfzdaprsample > jmfmz-darpsample > jmfzdaprsample > jmfmz-35mlw7mi7jcoa

jmfmz-35mlw7mi7jcoa

Event Hubs Namespace

Search (Ctrl+/,)

«

+ Event Hub

🔄 Refresh

Overview

Activity log

Access control (IAM)

Tags

Search to filter items...

Name	Status
dapr-eh	Active

Home > jmfzdaprsample > jmfmz-darpsample

jmfmz-darpsample

Service Bus Namespace

Search (Ctrl+/,)

«

+ Topic

🔄 Refresh

Overview

Activity log

Access control (IAM)

Tags

Search to filter items...

Name	Status
dapr	Active





# Ejemplo E2E

## Desarrollando la solución (1/11)

Debemos instalar DARP CLI:

<https://docs.dapr.io/getting-started/install-dapr-cli/>

Una vez seguidos los pasos, verifica que esta funcionando:

```
Símbolo del sistema
C:\dapr>dapr

dapr

=====
Distributed Application Runtime
Usage:
  dapr [command]

Available Commands:
  completion  Generates shell completion scripts
  components  List all Dapr components. Supported platforms: Kubernetes
  configurations  List all Dapr configurations. Supported platforms: Kubernetes
  dashboard   Start Dapr dashboard. Supported platforms: Kubernetes and self-hosted
  help        Help about any command
  init        Install Dapr on supported hosting platforms. Supported platforms: Kubernetes and self-hosted
  invoke      Invoke a method on a given Dapr application. Supported platforms: Self-hosted
  list        List all Dapr instances. Supported platforms: Kubernetes and self-hosted
  logs        Get Dapr sidecar logs for an application. Supported platforms: Kubernetes
  mtls        Check if mTLS is enabled. Supported platforms: Kubernetes
  publish     Publish a pub-sub event. Supported platforms: Self-hosted
  run         Run Dapr and (optionally) your application side by side. Supported platforms: Self-hosted
  status      Show the health status of Dapr services. Supported platforms: Kubernetes
```

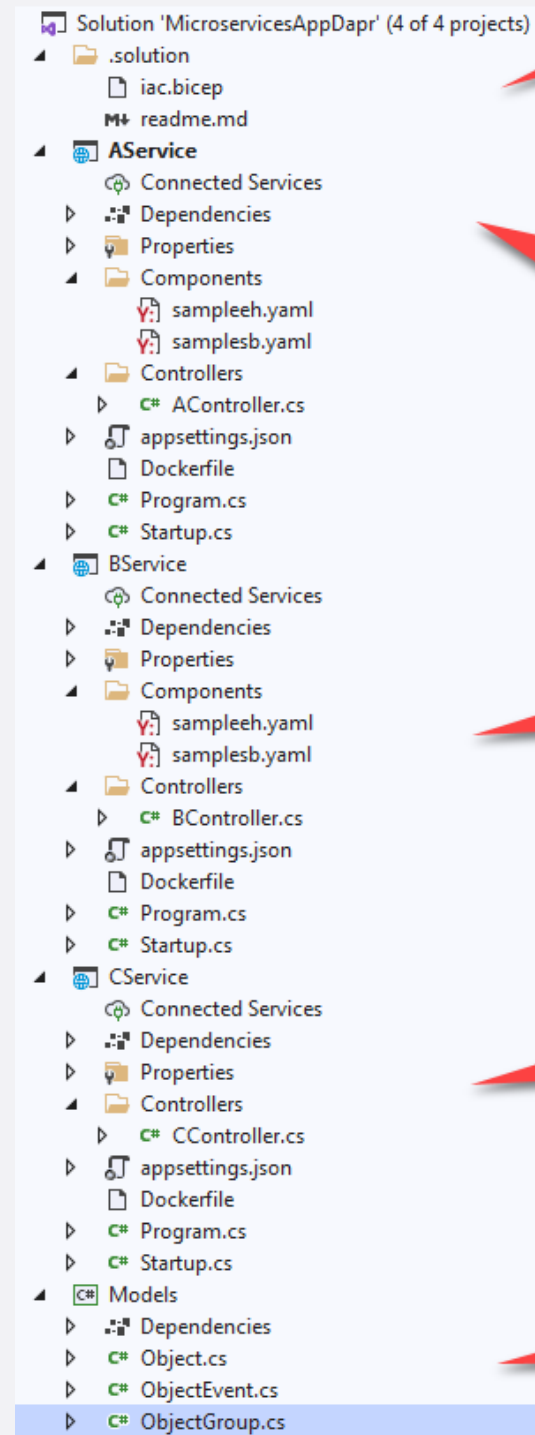
Inicializar con la versión 1.0.0, para que puedas ejecutar el workshop sin ningún problema:

```
C:\dapr>dapr init --runtime-version 1.0.0
Making the jump to hyperspace...
Downloading binaries and setting up components...
Downloaded binaries and completed components set up.
daprd binary has been installed to C:\Users\██████\AppData\Local\dapr\bin.
daprd container is running.
daprd_redis container is running.
daprd_zipkin container is running.
Use `docker ps` to check running containers.
Success! Dapr is up and running. To get started, go here: https://aka.ms/dapr-getting-started
```



# Ejemplo E2E

## Desarrollando la solución (2/11)



Infraestructura y Reame de la solución

Service A. Donde se encuentran los componentes de DAPR, el Servicio promiamente dicho y un archivo de Docker

Service B. Con una estructura similar al Service A.

Service C. No tiene componente.

Los modelos compartidos entre las soluciones

Una vez instaladas las herramientas de CLI de DARP es cuando comenzamos con nuestro ejemplo. Y para ello volvemos al proyecto que te has descargado de GitHub y carga la solución.

Los componentes en el código de ejemplo que has descargado necesitan conectar con la laC recientemente desplegada.

```
apiVersion: dapr.io/v1alpha1
kind: Component
metadata:
  name: daprsb
spec:
  type: pubsub.azure.servicebus
  metadata:
    - name: connectionString
      value: <yourconnectionstring>
```

Copia la cadena de conexión del Service Bus. Repite la operación con los demás fichero YAML alojados en las carpetas Component.

Supongo que estará familiarizado con Azure y por tanto localizar las cadenas de conexión te resultará algo trivial. En otro caso, como para cada tipo de recurso se encuentra en diferentes sitios, te recomiendo que busques en la documentación.



# Ejemplo E2E

## Desarrollando la solución (3/11)

Ya va siendo ahora de entrar en el código. Vamos a ir detallando las partes más importantes de la solución.

En los startups de cada servicio hemos añadido DARP al controlador mediante:

```
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllers().AddDapr();
}
```

Este código inyecta tanto el sidecard de DARP como el cliente en nuestros controladores. Para los servicios que usan el mecanismo PUB/SUB, debemos mapearlo:

```
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();
    app.UseCloudEvents();
    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapSubscribeHandler();
        endpoints.MapControllers();
    });
}
```



# Ejemplo E2E

## Desarrollando la solución (4/11)

Y usar la DI:

```
[ApiController]
3 references
public class BController : ControllerBase
{
    private readonly DaprClient _dapr;
    private readonly ILogger<BController> _logger;

    0 references
    public BController(ILogger<BController> logger, DaprClient dapr)
    {
        _logger = logger;
        _dapr = dapr;
    }
}
```

Una vez visto este punto común vamos a desgranar las partes más importantes de las distintas soluciones.



# Ejemplo E2E

## Desarrollando la solución (5/11)

El Service A, procesará un objeto y posteriormente publicará un evento de creación:

```
[HttpPost]
[Route( template: "CreateAS")]
0 references | Jose Maria Flores Zazo, 3 hours ago | 1 author, 1 change
public async Task<IActionResult> Create([FromBody] ObjectGroup og, [FromServices] DaprClient daprClient)
{
    _logger.LogInformation( message: $"Object Group with id {og.Id} created");
    await _dapr.PublishEventAsync<ObjectGroup>( pubsubName: "dapr-sb", topicName: "dapr", og);
    return Ok();
}
```

Creación con HTTP

```
[Topic( pubsubName: "dapr-sb", name: "dapr")]
[HttpPost]
[Route( template: "dapr")]
0 references | Jose Maria Flores Zazo, 3 hours ago | 1 author, 1 change
public async Task<IActionResult> Process([FromBody] ObjectGroup og)
{
    _logger.LogInformation( message: $"Object Group with id {og.Id} processed");
    await PublishEvent(og.Id, ObjectEvent.EventType.Created);
    return Ok();
}
```

Creación con Service Bus

Ambos publicarán el evento implicando a DARP. Como podrás observar en el código cada publicación se realiza de forma diferente para implementar la arquitectura anteriormente definida.

```
1 reference
private async Task<IActionResult> PublishEvent(Guid id, ObjectEvent.EventType type)
{
    var ev = new ObjectEvent
    {
        Id = id,
        Name = "ObjectEvent",
        Type = type
    };
    await _dapr.PublishEventAsync<ObjectEvent>( pubsubName: "dapr-eh", topicName: "CallBService", ev);
    return Ok();
}
```



# Ejemplo E2E

## Desarrollando la solución (6/11)

El Service B esta suscrito al Event Hub para activar las acciones correspondientes:

```
[Topic( pubsubName: "dapr-eh", name: "CallBService")]
[HttpPost]
[Route( template: "dapr")]
0 references | Jose Maria Flores Zazo, 3 hours ago | 1 author, 1 change
public async Task<IActionResult> ProcessEvent([FromBody] ObjectEvent ev)
{
    _logger.LogInformation( message: $"Received new event");
    _logger.LogInformation( message: "{0} {1} {2}", params args: ev.Id, ev.Name, ev.Type);

    switch (ev.Type)
    {
        case ObjectEvent.EventType.Created:
            if (await Get(ev.Id))
            {
                _logger.LogInformation( message: $"Created the Object Group {ev.Id}");
            }
            else
            {
                _logger.LogInformation( message: $"Object Group {ev.Id} not found in event Created");
            }

            break;
        case ObjectEvent.EventType.Updated:
            if (await Get(ev.Id))
            {
                _logger.LogInformation( message: $"Updated the Object Group {ev.Id}");
            }
            else
            {
                _logger.LogInformation( message: $"Object Group {ev.Id} not found in event Updated");
            }

            break;
        case ObjectEvent.EventType.Deleted:
            _logger.LogInformation( message: $"Deleted the Object Group {ev.Id}");
            break;
    }

    return Ok();
}
```



# Ejemplo E2E

## Desarrollando la solución (7/11)

En el código anterior usamos un tópico para decirle a DARP que use nuestro componente `dapr-eh.yaml` y que se suscriba al grupo de consumidores.

Y a continuación metemos la lógica del tipo de evento para realizar el proceso correspondiente.

También podrás observar un método `Get` que consulta al Service C si existe.

```
2 references
private async Task<bool> Get(Guid id)
{
    try
    {
        await _dapr.InvokeMethodAsync<object, ObjectGroup>(HttpMethod.Get, appId: "get", methodName: id.ToString(), data: null);
        return true;
    }
    catch (Exception)
    {
        //handle errors
    }

    return false;
}
```

Usamos una invocación asíncrona para llamar al Service C.



# Ejemplo E2E

## Desarrollando la solución (8/11)

Vamos a probar la solución.

Para ello vamos a necesitar entrar en la carpeta donde tenemos la solución y ejecutar una serie de comandos desde el CLI del sistema.

Nos aseguramos de tener DARP instalado, ya vimos como instalarlo anteriormente, por ejemplo:

```
Símbolo del sistema
C:\temp\microservicesappdapr>dapr init
Making the jump to hyperspace...
C:\Users\JoséMaríaFloresZazo\.dapr\bin\daprd.exe file already exists, please run `dapr uninstall` first before running `dapr init`
C:\temp\microservicesappdapr>
```

Y vamos a lanzar 3 símbolos de sistema, uno por cada servicio.





# Ejemplo E2E

## Desarrollando la solución (9/11)

En cada ventana del CLI vamos a tener corriendo cada servicio, para ello ejecutamos una instrucción cada vez en cada una de las carpetas de la solución:

- `dapr run --app-id aservice --components-path ./components --app-port 5002 dotnet run`
- `dapr run --app-id bservice --components-path ./components --app-port 5001 dotnet run`
- `dapr run --app-id cservice --app-port 5000 dotnet run`

Cuando lances las tres ventanas de CLI de forma independiente podrás ver algo similar a esto para cada una de ellas:

```

C:\> Símbolo del sistema - dapr run --app-id aservice --components-path ./components --app-port 5002 dotnet run
P-GSA2HFS scope=dapr.runtime.grpc.internal type=log ver=1.0.0
time="2021-03-22T18:07:56.460542+01:00" level=info msg="internal gRPC server is running on port 52210" app_id=aservice i
nstance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:07:56.4615395+01:00" level=info msg="application protocol: http. waiting on port 5002. This will bl
ock until the app is listening on that port." app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1
.0.0
Updating metadata for app command: dotnet run
You're up and running! Both Dapr and your app logs will appear here.

== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==      Now listening on: http://localhost:5002
== APP == info: Microsoft.Hosting.Lifetime[0]
== APP ==      Application started. Press Ctrl+C to shut down.
```

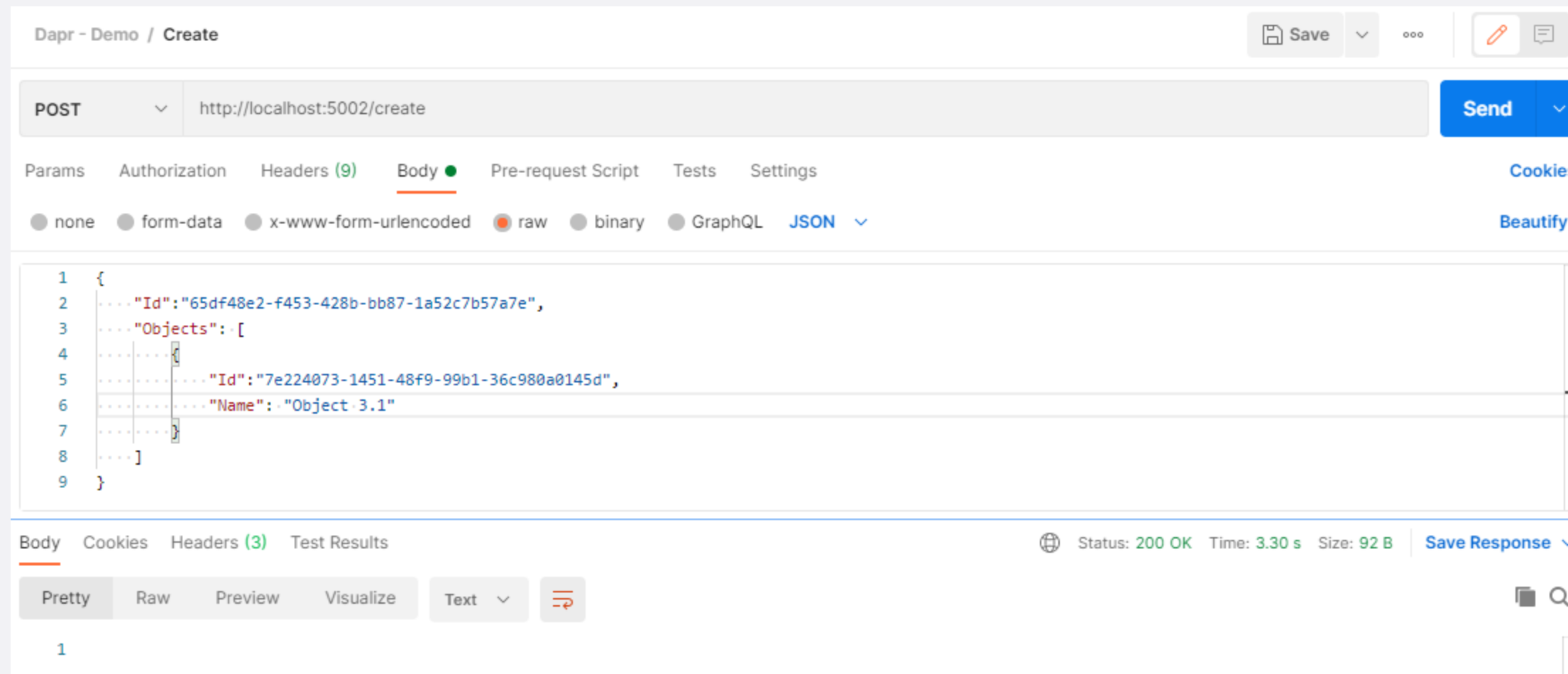
Si tienes conflictos con los puertos revisa el estado de tu máquina para usar otros puertos nuevos.



# Ejemplo E2E

## Desarrollando la solución (10/11)

Ahora es cuando vamos a lanzar una llamda HTTP via POSTMAN:



# Ejemplo E2E

## Desarrollando la solución (11/11)

Y podemos observar el comportamiento en los log:

```
Símbolo del sistema - dapr run --app-id aservice --components-path ./components --app-port 5002 dotnet run
Interval: 30s" app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime.actor type=log ver=1.0.0
time="2021-03-22T18:24:56.7544554+01:00" level=info msg="app is subscribed to the following topics: [dapr] through pubsub" app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:24:56.8484455+01:00" level=info msg="placement tables updated, version: 0" app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime.actor.internal.placement type=log ver=1.0.0
time="2021-03-22T18:24:57.6844988+01:00" level=info msg="dapr initialized. Status: Running. Init Elapsed 8470.8063ms" app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
== APP == info: AService.Controllers.OrderController[0]

== APP ==      Object Group with id 65df48e2-f453-428b-bb87-1a52c7b57a7e created

== APP == info: AService.Controllers.OrderController[0]

== APP ==      Object Group with id 65df48e2-f453-428b-bb87-1a52c7b57a7e processed
```

```
Símbolo del sistema - dapr run --app-id aservice --components-path ./components --app-port 5002 dotnet run
Interval: 30s" app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime.actor type=log ver=1.0.0
time="2021-03-22T18:24:56.7544554+01:00" level=info msg="app is subscribed to the following topics: [dapr] through pubsub" app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:24:56.8484455+01:00" level=info msg="placement tables updated, version: 0" app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime.actor.internal.placement type=log ver=1.0.0
time="2021-03-22T18:24:57.6844988+01:00" level=info msg="dapr initialized. Status: Running. Init Elapsed 8470.8063ms" app_id=aservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
== APP == info: AService.Controllers.OrderController[0]

== APP ==      Object Group with id 65df48e2-f453-428b-bb87-1a52c7b57a7e created

== APP == info: AService.Controllers.OrderController[0]

== APP ==      Object Group with id 65df48e2-f453-428b-bb87-1a52c7b57a7e processed

Símbolo del sistema - dapr run --app-id bservice --components-path ./components --app-port 5001 dotnet run
== APP == info: Microsoft.Hosting.Lifetime[0]

== APP ==      Now listening on: http://localhost:5001

== APP == info: Microsoft.Hosting.Lifetime[0]

== APP ==      Application started. Press Ctrl+C to shut down.

== APP == info: Microsoft.Hosting.Lifetime[0]

== APP ==      Hosting environment: Development

== APP == info: Microsoft.Hosting.Lifetime[0]

== APP ==      Content root path: C:\temp\microservicesapp\dapr\BService

time="2021-03-22T18:25:02.9241497+01:00" level=info msg="application discovered on port 5001" app_id=bservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:25:03.083146+01:00" level=info msg="application configuration loaded" app_id=bservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0

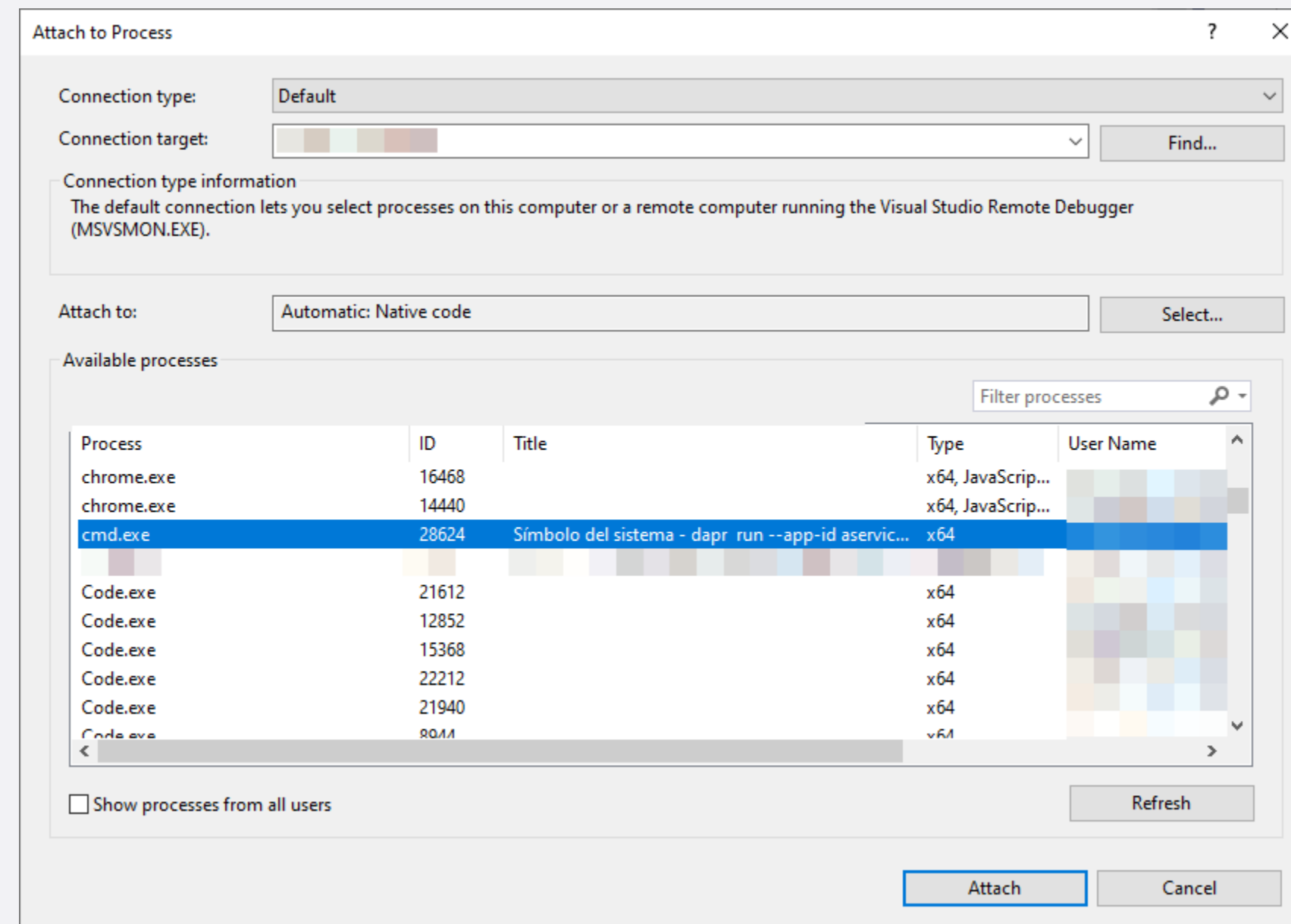
Símbolo del sistema - dapr run --app-id cservice --app-port 5000 dotnet run
time="2021-03-22T18:24:59.2077829+01:00" level=info msg="http server is running on port 54878" app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:24:59.2087819+01:00" level=info msg="The request body size parameter is: 4" app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:24:59.2097884+01:00" level=info msg="enabled gRPC tracing middleware" app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime.grpc.internal type=log ver=1.0.0
time="2021-03-22T18:24:59.2163441+01:00" level=info msg="enabled gRPC metrics middleware" app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime.grpc.internal type=log ver=1.0.0
time="2021-03-22T18:24:59.2173389+01:00" level=info msg="internal gRPC server is running on port 54891" app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:24:59.2183384+01:00" level=info msg="application protocol: http, waiting on port 5000. This will block until the app is listening on that port." app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:24:59.233794+01:00" level=info msg="application discovered on port 5000" app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:24:59.3750667+01:00" level=info msg="application configuration loaded" app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime type=log ver=1.0.0
time="2021-03-22T18:24:59.3810528+01:00" level=info msg="actor runtime started. actor idle timeout: 1h0m0s. actor scan interval: 30s" app_id=cservice instance=DESKTOP-GSA2HFS scope=dapr.runtime.actor type=log ver=1.0.0
Updating metadata for app command: dotnet run
You're up and running! Both Dapr and your app logs will appear here.
```



# Ejemplo E2E

## Depuración de la solución

Una de las formas es enlazar el proceso:



# Ejemplo E2E

## Azure API Management <sup>(1/7)</sup>

Vamos a ver como integrar DAPR con Azure API Management. Dado que esto involucra muchos sistemas diferentes y que en realidad no es el objeto de este workshop, voy a mostrar que puede ofrecernos APIM (API Management) en el ámbito de DAPR, no voy a explicar un paso a paso para la integración con DAPR, esto te lo dejo a ti.

¿Qué necesitamos?

- **K8s Cluster**, ya sea AKS, Minikube o Docker Desktop's K8s.
- **API Management**, en este caso con self-hosted API gateway en nuestro cluster. Para más información ver: <https://docs.microsoft.com/en-us/azure/api-management/api-management-howto-provision-self-hosted-gateway>
- **DAPR**, debemos instalarlo en nuestro cluster y para ello os dejo el siguiente enlace, donde se muestra como instalarlo en un K8s: <https://github.com/dapr/cli#install-dapr-on-kubernetes>
- Azure Service Bus, para nuestro servicio PUB/SUB.

Aquí existe mucha información que debes poner en práctica, como decía antes, el objetivo es que vea la importancia de APIM en el ecosistema y como podrás hacer soluciones más robustas sin escribir nada de código.

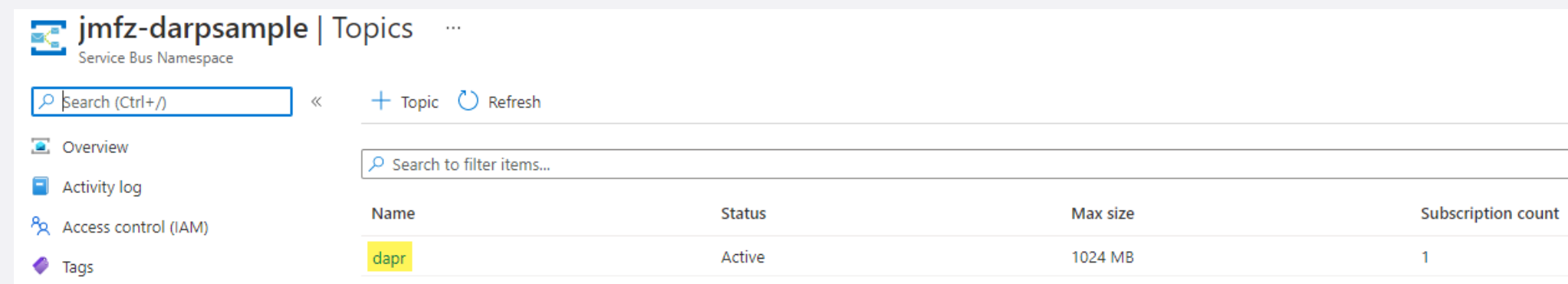
El deber como arquitecto es evitar que los desarrolladores reinventen la rueda. Y como desarrollador, siempre es bueno conocer estas piezas y como encajan.



# Ejemplo E2E

## Azure API Management (2/7)

Para continuar con nuestro ejemplo, vamos a usar el t3pico:



**jmfz-darpsample | Topics** ...

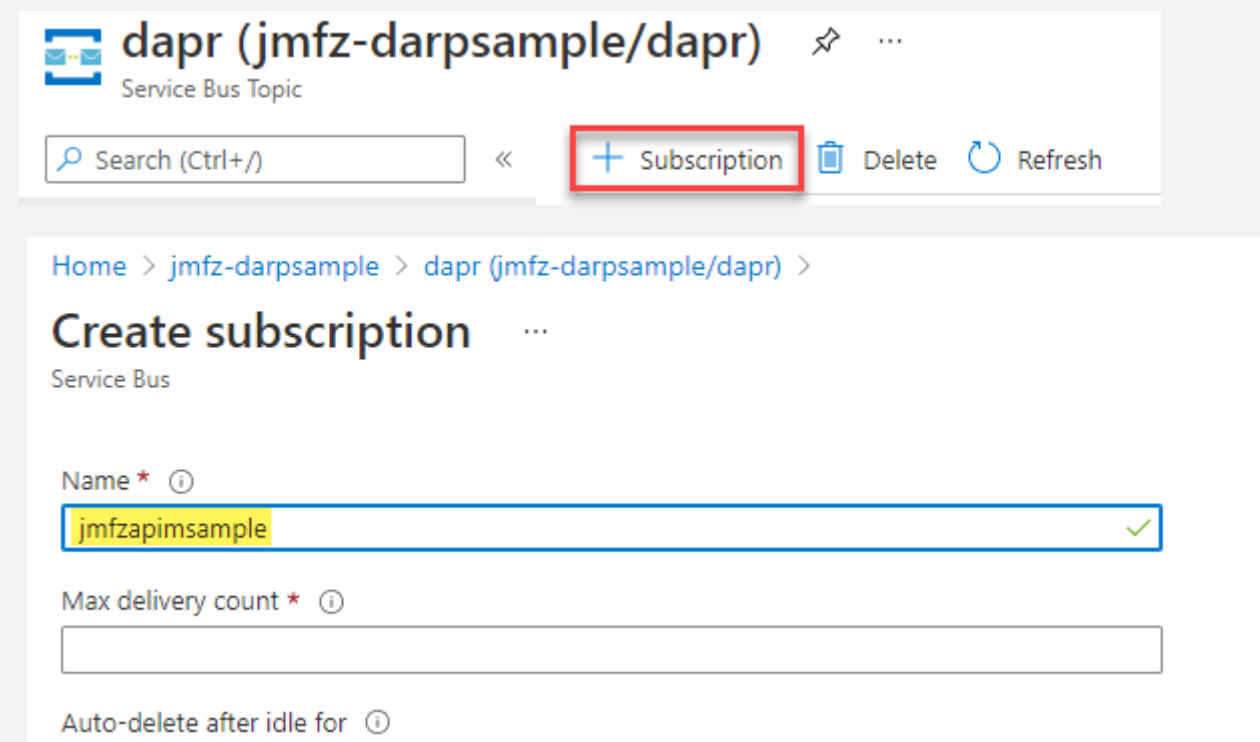
Service Bus Namespace

Search (Ctrl+/) << + Topic Refresh

Search to filter items...

Name	Status	Max size	Subscription count
dapr	Active	1024 MB	1

Y un nuevo suscriptor:



**dapr (jmfz-darpsample/dapr)** ...

Service Bus Topic

Search (Ctrl+/) << + Subscription Delete Refresh

Home > jmfz-darpsample > dapr (jmfz-darpsample/dapr) >

### Create subscription

Service Bus

Name \* ⓘ

jmfzapimsample ✓

Max delivery count \* ⓘ

Auto-delete after idle for ⓘ



# Ejemplo E2E

## Azure API Management (3/7)

Instala Dapr en el cluster:

```
Símbolo del sistema

C:\temp\microservicesappdapr>dapr init -k -n jmfzapinsample
Making the jump to hyperspace...
Note: To install Dapr using Helm, see here: https://docs.dapr.io/getting-started
Deploying the Dapr control plane to your cluster...
Success! Dapr has been installed to namespace jmfzapinsample. To verify, run
C:\temp\microservicesappdapr>_
```

Y un nuevo componente:

```
sample-sb2.yaml  AController.cs  readme.md  Progra
metadata
1  apiVersion: dapr.io/v1alpha1
2  kind: Component
3  metadata:
4    namespace: jmfzapinsample
5    name: dapr-sb
6  spec:
7    type: pubsub.azure.servicebus
8    version: v1
9    metadata:
10 - name: connectionString # Required
11   value: "Endpoint=sb://jmfz-darpsample.servicebus.windows.net;"
```



# Ejemplo E2E

## Azure API Management (4/7)

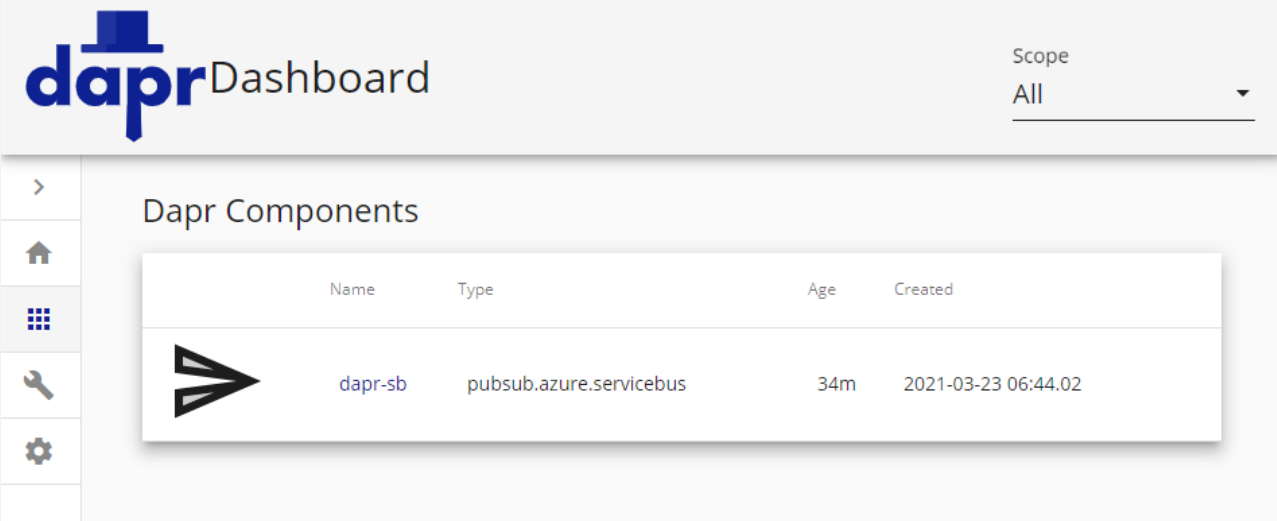
El componente contiene secretos, podemos usar otras técnicas para ocultarlos, pero no es el objetivo del ejercicio.  
Puedes ver más información en: <https://docs.dapr.io/operations/components/component-secrets/>

Vamos a añadir nuestro cluster:

```
C:\temp\microservicesappdapr>kubectl apply -f sample-sb2.yaml
component.dapr.io/dapr-sb created
```

Y si lo deseas entrar en el Dashboard:

```
C:\temp\microservicesappdapr>dapr dashboard -k -n jmfzapinsample
Dapr dashboard found in namespace:      jmfzapinsample
Dapr dashboard available at:      http://localhost:8080
```



The screenshot shows the Dapr Dashboard interface. At the top, there's a header with the 'dapr' logo and the word 'Dashboard'. On the right, there's a 'Scope' dropdown menu set to 'All'. Below the header, there's a sidebar with navigation icons: a chevron, a home icon, a grid icon, a key icon, and a gear icon. The main content area is titled 'Dapr Components' and contains a table with the following data:

Name	Type	Age	Created
dapr-sb	pubsub.azure.servicebus	34m	2021-03-23 06:44:02





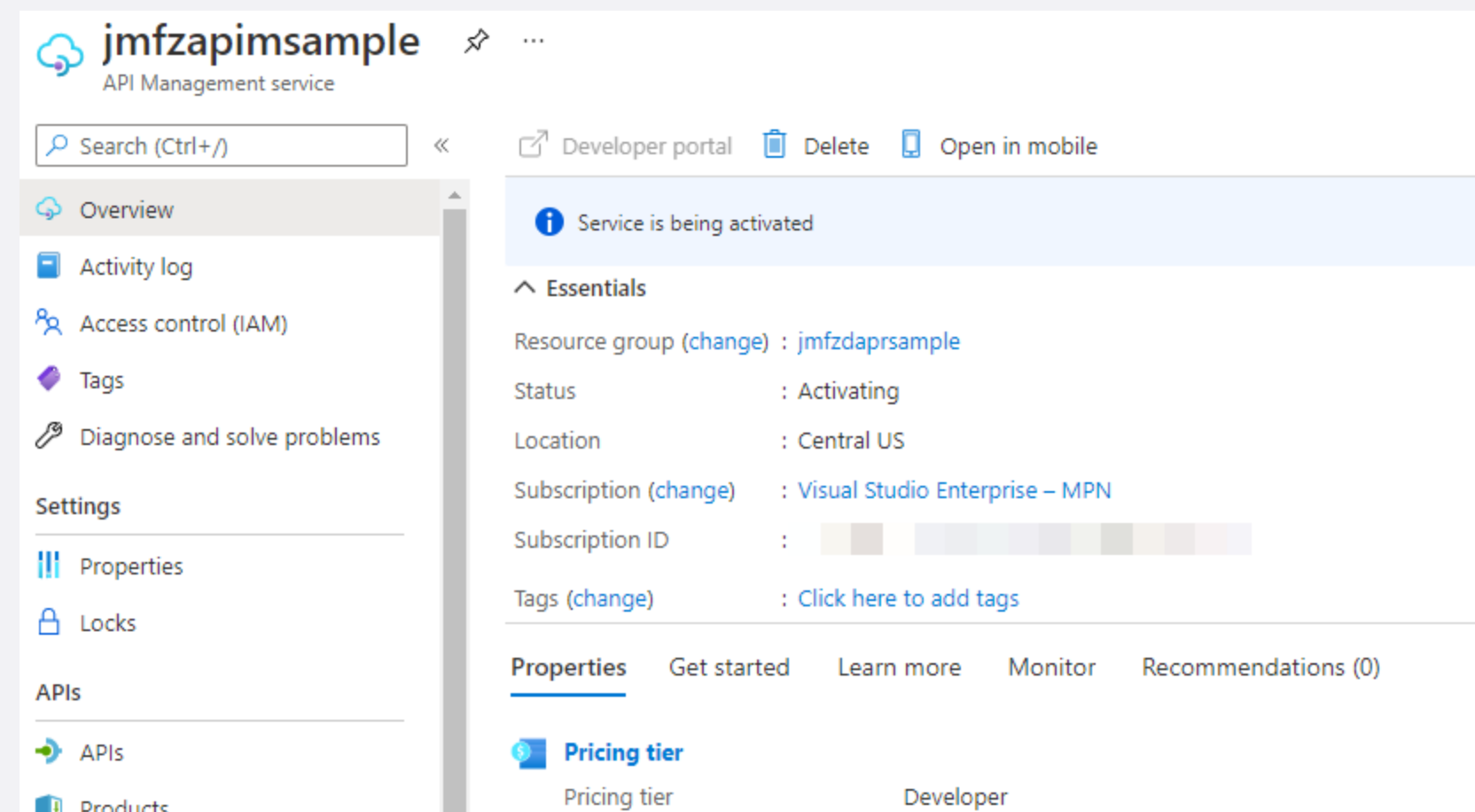
# Ejemplo E2E

## Azure API Management <sup>(5/7)</sup>

El siguiente paso es configurar el self-hosted API gateway para K8s:

<https://github.com/MicrosoftDocs/azure-docs.es-es/blob/master/articles/api-management/how-to-deploy-self-hosted-gateway-kubernetes.md>

Creamos nuestro Azure API Management (APIM):



# Ejemplo E2E

## Azure API Management (6/7)

Añadimos el enlace al API:

Create a blank API

Basic | **Full**

\* Display name

\* Name

Web service URL

API URL suffix

Base URL

Creamos nuestro enlace contra el endpoint:

REVISION 1 CREATED Mar 23, 2021, 9:14:31 AM

Design Settings Test Revisions Change log

Search operations

Filter by tags

Group by tag

+ Add operation

All operations

No operations to display.

jmfzapimsample > Add operation

Frontend

\* Display name

\* Name

\* URL

Description

Tags



# Ejemplo E2E

## Azure API Management (6/7)

Es obligatorio establecer policy para nuestro Service Bus:

<https://docs.microsoft.com/es-es/azure/api-management/api-management-dapr-policies>

XML Copiar

```
<policies>
  <inbound>
    <base />
    <publish-to-dapr
      pubsub-name="orders"
      topic="new"
      response-variable-name="dapr-response">
      @(context.Request.Body.As<string>())
    </publish-to-dapr>
  </inbound>
  <backend>
  </backend>
  <outbound>
    <base />
  </outbound>
  <on-error>
    <base />
    <return-response response-variable-name="pubsub-response" />
  </on-error>
</policies>
```



# Ejemplo E2E

## Azure API Management (7/7)

Y para poder hacer finalmente uso de este APIM, es obligatorio usar la key de suscripción:

POST

https://jmfzapimsample.azure-api.net/sample-sb/create

Send

Params

Authorization

Headers (11)

Body

Pre-request Script

Tests

Settings

Cookies

<input checked="" type="checkbox"/>	User-Agent ⓘ	PostmanRuntime/7.26.10	
<input checked="" type="checkbox"/>	Accept ⓘ	*/*	
<input type="checkbox"/>	Accept-Encoding ⓘ	gzip, deflate, br	
<input checked="" type="checkbox"/>	Connection ⓘ	keep-alive	
<input checked="" type="checkbox"/>	Ocp-Apim-Subscription-Key		
<input checked="" type="checkbox"/>	Ocp-Apim-Trace	True	
	Key	Value	Description

A partir de aquí ya podrás llamar al A Service con las policitas correspondientes y la key.

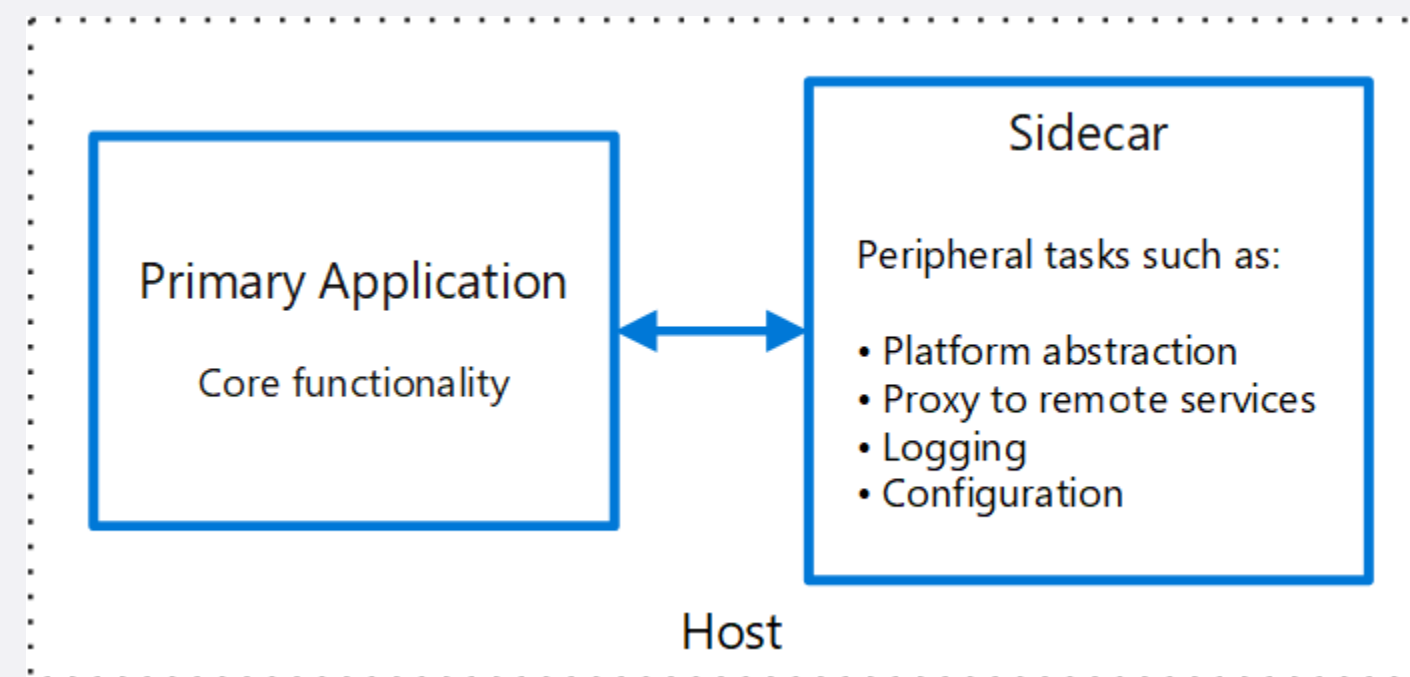


# Bonus

## Patrón Sidecar

**SIDECAR** — *(RAE)* Asiento lateral adosado a una motocicleta y apoyado en una rueda

Tal y como viste en el diagrama del principio donde aparecía el concepto de Sidecar, podrás deducir que el nombre que han puesto al patrón esclarece mucho su funcionalidad.



En este enlace tenéis una buena explicación de este patrón:

<https://docs.microsoft.com/es-es/azure/architecture/patterns/sidecar>





**¡Gracias!**



Puedes encontrarme buscando por **jmfloreszazo** en

